📌 2. Requirement Analysis Phase (Detailed)

In this phase, we identified and categorized the essential **functional** and **non-functional** requirements of the **Freelancing Application**, ensuring that the platform supports a smooth and secure user experience while achieving its core objectives.

📩 Functional Requirements (What the system must do)

Functional requirements describe the **core features** and actions that the system must be able to perform for users. These form the **backbone of the user journey**.

1. User Authentication (Register/Login)

- The system must allow users to register with necessary credentials (name, email, password, role).
- During login, credentials must be verified, and a secure token (JWT) should be issued for authenticated access.
- Passwords must be securely hashed (using bcrypt or similar) to ensure data protection.
- After login, users should access features based on their assigned roles.

2. Role-Based Access Control (Client, Freelancer, Admin)

- The application supports three roles:
    - **Client** – Can post jobs, view bids, select freelancers, and give feedback.
    - **Freelancer** – Can view projects, place bids, and submit completed work.
    - **Admin** – Has privileges to view, block/unblock users and monitor platform activity.
- Certain pages and features must be restricted based on the user's role to ensure controlled access.

3. Project Posting and Viewing

- A client can **create a project** by filling out a form with a title, description, and optional budget.
- Once submitted, the project is stored in the database and made publicly available for freelancers to view.
- Freelancers must be able to view all available projects with essential details and category filters.

4. Bid Placement and Viewing

- Freelancers can **place bids** on projects by entering the proposed price and a short proposal/message.
- Clients can **view all bids** received for their project in a tabular/list view, showing freelancer names and proposal content.
- Clients can **select one freelancer** to assign the project, after which the selected freelancer can begin work.

5. Work Submission

- After a client selects a freelancer, the freelancer should be able to **submit completed work** through a form or file upload.
- The submission must be linked to the project and stored in the database.
- Clients can then mark the work as complete and proceed to the feedback phase.

6. Feedback and Rating

- Once the freelancer submits work and it is accepted by the client, the client should be able to **rate the freelancer (1 to 5 stars)** and leave a brief comment.
- This feedback is stored and linked to the freelancer's profile, allowing others to assess their reputation.
- The platform must be able to calculate and display **average ratings** for each freelancer based on past reviews.

7. Admin Panel – User Management

- Admin should have access to a secure dashboard to view all registered users.
- Admin can **block or unblock users**, which disables or re-enables their access to the platform.
- Admin should not be able to delete users permanently but should manage access for safety.
- Admin actions must be logged and validated using token-based authentication.

🔐 **Non-Functional Requirements (How the system behaves)**

Non-functional requirements focus on **performance, usability, and quality** of the system rather than features.

1. Responsive Design

- The frontend must be fully responsive using CSS frameworks like **Tailwind CSS** or **Bootstrap**.
- It should work on all devices — desktop, tablet, and mobile — with a user-friendly layout for each screen size.
- UI components like navbars, buttons, forms, and tables should adapt smoothly.

2. Secure Authentication Using JWT

- All protected routes (backend APIs) must be secured using **JSON Web Tokens**.
- JWTs must be stored securely on the frontend (usually in memory or HttpOnly cookies).
- Middleware should validate tokens before processing any request.
- Role-based access must also be enforced in backend controllers.

3. Deployment and Hosting (Render)

- Both frontend and backend should be deployed using Render or any similar platform.
- Environment variables (e.g., MongoDB URI, JWT_SECRET) should be securely stored and used.
- The deployed version should be publicly accessible for demo purposes.

4. Fast API Response & Error Handling

- All API routes must respond within a few hundred milliseconds for common operations.
- Every controller should use proper **try-catch blocks** to gracefully handle errors.
- Client-side error messages should be clean, user-friendly, and indicate next steps (e.g., “Invalid login credentials”).

📊 **Summary Table**

Requirement Type	Description
Functional	Register/Login, Role-based access, Project posting, Bidding, Feedback, Admin controls
Non-Functional	Responsive UI, JWT security, Deployment readiness, Fast response time, Error handling